

PRAGMA'S

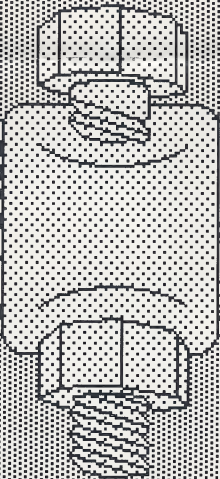
Distributed
FREE
to over 4,000 Pick installations!

PRODUCT PROFILES

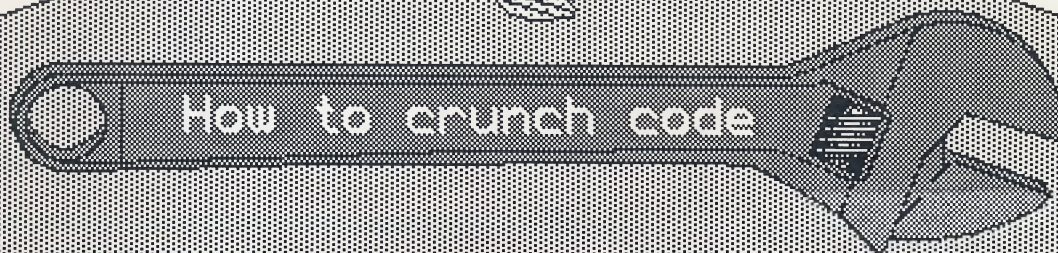
News and Information for Pick™ Operating System Users

Issue Number 29

August 1986



```
NEW LABEL  
PAST.COL =  
LEFT.PART = STARTC  
RIGHT.PART = SOURCE  
SOURCE.LINE =  
COLUMN = STARTC  
IF TOKEN = 2 TH  
IF PASS = 2 TH  
+ LEN(TOKEN)  
NE(1, STARTC)  
LINE(PAST.COL, 1)  
AT:NEW.LINE  
LEN(NEW.LINE)  
CTN.TYPE = 3  
ITEM = ""
```



207 GRANADA DRIVE
APTOS, CA 95003

BULK RATE
U.S. POSTAGE
PAID
APTOS, CA 95003
Permit No. 67

Address Correction Requested

Pragma's PRODUCT PROFILES

Issue #29 • August 1986

Product Profiles is published periodically by:

Semaphore Corporation
207 Granada Drive
Aptos, CA 95003

Telephone 408-688-9200

Entire contents copyright © 1986 by Semaphore Corporation • All rights reserved • No part of this publication may be reproduced in any form or by any means without prior written permission from Semaphore Corporation • Semaphore offers no warranty, either express or implied, for any losses due to the use of any material published in *Product Profiles* • Subscriptions are free, but are available only to Pick operating system users in the USA • Back issues are available by sending a stamped, self-addressed envelope • All material received will be considered for publication • Semaphore reserves the right to edit all submittals • Please tell us your mailing label number when calling or writing • Semaphore cannot be responsible for missing or lost issues • Send your new address at least four weeks before moving • Pick is a trademark of Pick Systems • Semaphore Corporation and its publications are not affiliated with Pick Systems.

Does Your Mailing Label Say RENEW?

If the mailing label on this issue says **RENEW**, your free subscription has expired. To avoid missing the next issue, renew immediately by returning this coupon!

Name: _____

Company: _____

Address: _____

City/State/ZIP: _____

Telephone: _____

I use the Pick operating system and my mailing address is in the USA. Please continue sending me *Pragma's Product Profiles* free of charge.

Signature: _____

Date: _____

Return this form to:
Pragma • 207 Granada Drive • Aptos, CA 95003

How to crunch code

Whenever a BASIC program's source code is compiled, the resulting object code is placed in sequential disk frames. When the object code is executed, BASIC's runtime interpreter scans and processes the object code byte by byte, starting with the first frame of object code for the program. Whenever the interpreter accesses a given frame of object code, and the frame isn't already in memory, a *frame fault* occurs. That is, the running program effectively stops and waits for the operating system to bring the required frame into memory from disk, so that the interpreter can continue. Because mechanical disks are rather slow, frame faults can drastically degrade system performance.

Minimizing frame faults means maximizing system throughput. That implies that it is desirable to "crunch code", or keep object code squeezed into as few disk frames as possible. The fewer frames a given BASIC program occupies, the fewer frame faults it will cause, while more memory will be available for other programs trying to run at the same time.

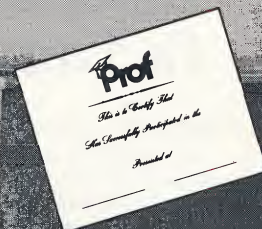
One quick and easy way to crunch code is to use the (C) option when compiling. (The compiler features and results discussed here may vary slightly for your machine. Consult your compiler's documentation to find out exactly what features are available on your system.) To allow runtime error messages and the debugger to reference line numbers, the compiler normally generates one extra byte of object code for each line of source code. The (C) option tells the compiler to suppress those extra end-of-line bytes. That means a 500-line program will save 500 bytes of object code, and occupy one less frame of memory. The disadvantage is that line numbers won't appear in error messages or be available to the debugger, but programmers shouldn't use the (C) option on undebugged programs anyway.

The compiler's (M) map option is another helpful code crunching tool, in that it can indicate when crunching is worth the effort. The end of the map that is output when (M) is used shows what frames of object code are generated for what lines of source code. If the last frame of object code represents 20 source code lines, it might be hard to crunch enough code to squeeze away a whole frame. But if only the last two or three source code lines overflowed into one whole object frame, there's a good chance enough bytes can be squeezed out so that the object code shrinks enough to make the last (mostly empty) frame drop off. Always compile with the (M) option to determine if an attempt at crunching code has actually saved any disk frames.

After using the compiler's (C) option, crunching object code becomes much more difficult, and requires knowing what kind of object code is generated for a

GET AN EDUCATION!

Now you can get an education on PICK® at home, in the office or anywhere you've got a PICK-based computer system and a CRT terminal.



Higher Education with PROF Software.

No longer will you be tardy at understanding the PICK database, TCL, EDITOR, ACCESS, PROC, and BASIC. With the PROF electronic learning system you can master the fundamentals of PICK on your own, at your own pace. PROF can be used *on site* over and over again by DP Managers to train new employees without the use of costly seminars. A complete PICK education is now available right on your computer system.



PROF leads you through simulated, hands-on work sessions in which you actively participate. You try every important command and carefully review the way the system responds to each. It all happens right on the terminal. There is no book to hold in your lap.

PICK is a registered trademark of PICK systems, Irvine, California
Prime INFORMATION is a registered trademark of Prime Computer, Inc.

The time to get an education is now! You can order PROF for your PICK, Microdata, or Prime INFORMATION™ system by writing or calling us at:

CRESCENDO Associates, Inc.

24350 Joy Road, Suite 9 Redford Township, Michigan 48239 (313) 537-1919

YES! Please send me more information about PROF.

Name

Company

Address

City State Zip

Phone ()

Computer System

Please fill out or attach business card and return.

Unlock The Secrets In Your Computer!

Pragma (not to be confused with *Pragma's Product Profiles*) is the original 48-page technical journal for Pick users published quarterly beginning in August 1982. Each issue is packed with software and helpful information, including complete and debugged program listings and detailed, explanatory articles for readers at all levels of experience. Order your **Pragma** issues today and begin unlocking the secrets in your Pick system!

Pragma #1: Welcome to Pragma ° ZIP Code File Design ° A Program for Renumbering Sheets ° Dealing with Data Counts ° VANILLA, The No-Frills Manufacturing System: The User's Manual ° More Memory, Fewer Reads ° SYSMAP: A Cross-Reference System ° How to Flag Missing Checks ° 23 Wish List Items ° Comparing Macroware with ENGLISH ° Building ENGLISH Headings with Queries ° Making the Compiler Hash Well ° Tape Handling ° Patching the Fault in Version 3.2 ° CANCEL PROC ° The Shell Game.

Pragma #2: We Have Liftoff ° A Modulo Setting Program ° Black Box Formatting ° TRIM.DELIM and PROFILE Utilities ° SYSMAP, A Cross-Reference System: File Format Input ° Galinski Hamburg User Profile ° LOOP vs. LOCATE Benchmarks ° 25 Wish List Items ° An Introduction to ENGLISH: Jargon ° Proc Conversions ° VANILLA, The No-Frills Manufacturing System: Bill of Material Input ° 14 Queries ° The Trouble Tree ° 2 Letters ° A Switchbox for 32 Ports ° Security and the DATA/BASIC Programmer ° The Cookie Game ° Some New Subscribers.

Pragma #3: Is Pragma a Rare Medium, Well Done? ° Edit Aides ° A Proc for Cross-Referencing Q-Pointers ° SYSMAP, A Cross-Reference System: Remaining File Input ° Rainbow Natural Foods User Profile ° More BASIC Benchmark Comparisons ° Justifying Ragged Output ° 13 Wish List Items ° Generating Monthly Column Headings ° VANILLA, The No-Frills Manufacturing System: Purchasing ° 5 Queries ° An Introduction to ENGLISH: More Commands ° Shared Site Checklist ° Converting Paint to Programs ° 3 Letters ° Generating Blank Forms ° Animal, A Game That Learns ° More New Subscribers.

Pragma #4: Survey Says... ° Pacific Valley Bank User Profile ° SYSMAP, A Cross-Reference System: Dicts and Procs ° Uncompiling: Unassembling Stack Code ° Left vs. Right Justification Benchmarks ° 4 Wish List Items ° A Comparison of BASIC Implementations ° More New Subscribers ° An Undocumented Editor Capability ° 3 Local User Group Reports ° Avoiding Saved Lists with PQ-RESELECT ° Self-Documenting Reports ° A Query ° A Survey of Hardware that Supports Pick-Style Software ° Printer Trade-Offs ° An Introduction to ENGLISH: Finding Files ° A Letter ° VANILLA, The No-Frills Manufacturing System: Purchase Order Entry ° The Swat! Game.

Pragma #5: Happy Birthday! ° Interactive Systems Producer Profile ° Uncompiling: Regenerating Source Code ° A Day of Revelation ° IBM Personal Computer vs. Microdata Reality Benchmarks ° VANILLA, The No-Frills Manufacturing System: Receiving ° 3 Wish List Items ° An Introduction to ENGLISH: Being Choosy ° Converting Manual Paint to Programs ° 6 Local User Group Reports ° More New Subscribers ° A Program That Reports File Pointer Locations ° Boiler Plate Processing with Runoff ° A New Query and an Old Query Answered ° SYSMAP, A Cross-Reference System: Automation ° Tape Types ° 6 Letters ° Permuted Index to the First Four Issues of Pragma ° Amazing, a Maze Game.

Pragma #6: Pick Pie Pictured ° The Ubiquitous POINTER-FILE ° Uncompiling: Resolving Labels ° An Introduction to ENGLISH: Syntax Overview ° AccuSoft Producer Profile ° Rounding Out 7 Benchmarks On 5 Machines ° Designing Data Entry Programs ° 12 Wish List Items ° GET: An Input Processor ° 5 Local User Group Reports ° More New Subscribers ° Do You Know Your Proc Limits? ° VANILLA, The No-Frills Manufacturing System: Inspection ° 2 Queries ° Individual Accounts or Shared Accounts? ° Lock Logic Illustrated ° Password Protection ° Two Undocumented Conversions ° A Letter ° How AMAZING Works.

Pragma #7: More New Subscribers ° A New Machine Visits Pragma ° Bantam Hardware Overview ° Suppressing LOCKED Clauses ° An Introduction to ENGLISH: WITH, BY and TOTAL ° Bantam Producer Profile ° VANILLA, The No-Frills Manufacturing System: Disposition ° A Bantam Diary ° New Benchmark Timings for 8 More Machines ° 6 Local User Group Reports ° GET: Source Code ° A Preprocessor for Symbolic Statement Labels ° 2 Wish List Items ° 3 Queries ° Bantam Software Overview ° A Letter ° The Slide Game.

Send \$25 for each 48-page back issue to:

Pragma
207 Granada Drive
Aptos, CA 95003

given piece of source code. The tool that helps do that is the compiler's (A) option, which outputs object code in mnemonic form. If your system doesn't offer the (A) option, you can use an "uncompiler" like the one featured in *Pragma* #4. Note that although the (A) option displays the various object code stack operators in hexadecimal, individual object code instruction lengths are usually not shown, so code crunchers must also be prepared to pick apart object code with verbs like DUMP. Nevertheless, compiling small test programs with (A) and then DUMPing the object code can reveal all kinds of ways to crunch code, which is how the examples presented below were discovered.

One way to crunch code is to use OCONV instead of FIELD. For example, the statement `X = FIELD(X, "*", 5)` generates 17 bytes of object code with our compiler, while `X = OCONV(X, "G5*1")` generates only 13 bytes. However, FIELD is typically much faster than OCONV, as reported in *Pragma* #3. (Code crunchers soon learn that saving space is almost always a trade-off for execution time.) Also, if the third argument to FIELD is a variable instead of a constant, it will compile to only 13 bytes just like OCONV!

FIELD's object code shrinks with variable arguments because the compiler generates six bytes of inline code for a numeric constant but uses only two bytes for a variable reference. That means that statements like `X=33` require ten bytes of code (one byte for the interpreter's LOAD opcode, six bytes for the value 33, one byte for the STORE opcode, and two bytes for the address of X), while `X=Y` only needs six bytes (three bytes each for a variable LOAD and variable STORE). So if a number is going to be used more than twice, assign it to a variable and use the variable name instead. For example, `X=3.14; Y=3.14; Z=3.14` requires 30 bytes of code, but `PI=3.14; X=PI; Y=PI; Z=PI` requires only 28 bytes. However, numbers like 0 and 1 are handled by the compiler as special cases, so the statements `X=0` and `X=1` require only four bytes each.

While numeric constants require six bytes of code, string constants only occupy their actual character length plus a special end-of-string byte. So while `X=33` requires ten bytes, `X="33"` requires only seven bytes (a one-byte LOAD opcode, the two-byte string, one end-of-string byte, and a three-byte STORE). However, `X="33"` saves X internally as a string instead of a number, which may require more storage space for X in some machines. Interestingly, the statement `X="33"+0` not only saves X as a number just like `X=33`, it also generates one less byte of object code!

Because compiled string constants vary with their length, all kinds of trade-offs begin to appear when strings are involved. For example, `PRINT <string>` only requires three bytes plus the number of characters in <string>, so a statement like `PRINT "%"` only needs four bytes

each time it's used in a program. But the statement `PRINT "PERCENT"` uses ten bytes, so if that statement is used repeatedly, it's better to first use `X="PERCENT"` (twelve bytes) and then `PRINT X` (four bytes) each time. A similar example is `PRINT @(79,23):`, which generates 16 bytes. If that cursor position appears in lots of `PRINT` statements, use `CORNER= @(76,23)` for 18 bytes, then `PRINT CORNER:` for four bytes each time.

The `SPACE` and `STR` functions also offer alternatives to string constants. The statement `PRINT SPACE():` with any numeric constant argument between the parentheses will generate only nine bytes of object code, so use statements like `PRINT " ":` for strings up to five blanks long, and use `SPACE():` for longer runs. Similarly, `PRINT STR(" ",10):` generates 12 bytes of code (and always 12, whenever `STR`'s second argument is a constant), so use `PRINT "****":` and the like for strings up to eight characters long, and `STR():` for longer runs of characters.

WANTED

Telemarketing software for Microdata Sequel 9000
Call (612) 861-4000 or write Metro Sales, Attn: Sandy
1620 East 78th Street, Richfield, MN 55423

ZEBRA

HARDWARE & PERIPHERALS

SO CAL FIELD SERVICE
WE BUY USED ZEBRAS

ATKIN/JONES
COMPUTER SERVICE

(714) 953-4351

Pick™
MAILING LISTS

P/Mail by Marianne
1893 Nadina Street
Seaside, CA 93955

Microdata™ Systems & Upgrades

Reality

128 MB Reflex II
50 MB Reflex I
128 K Mos Memory
Intelligent 8-Ways
Tape Drives
Complete Systems

Sequel

IMB Memory
ACLC
Disc Drives
Controllers
Complete Systems

- Quick Delivery
- Competitive Prices
- Trade-Ins Accepted
- Professional Service
- Repair
- New Or Used
- Maintenance Guarantee
- Adds, Wyse Crts

Printronix Printers

BUY SELL TRADE

EASTCOMP II INC.

CALL CHUCK HAAS
513-528-5400

MICRODATA EQUIPMENT FOR SALE

(1) Sequel 9000 VMS System

- Base system
- Possibility of selling manufacturing software with base system

Miscellaneous

- 1 256 MB Fujitsu Disc
- 1 MB Memory
- 4 ACLC
- 1 5750 Communications Terminal
- 5 Prism IV Terminals
- 2 Microdata Reality Systems
 - Make offer

Contact: Priscilla Cain
Flow Systems, Inc.
(206) 872-4900

FREE Back Issues!

A few back issues of *Pragma's Product Profiles* are still available while supplies last. To receive your FREE copies, indicate which issues you need and send a stamped, self-addressed envelope to:

Pragma • 207 Granada Dr. • Aptos CA 95003

(Allow 1.5 oz. per issue to compute postage.)

- Issue #10:** Beyond The Power Of The Pick System
- Issue #11:** Open Architecture Status Report
- Issue #12:** Our Impressions Of The Zebra 750
- Issue #14:** Our Third Annual Pick Hardware Survey
- Issue #15:** The Wonders Of A Software Upgrade
- Issue #17:** A READ Sometimes Fails
- Issue #19:** New Pick Book A Disappointment
- Issue #21:** Programming For Speed
- Issue #22:** How Files Grow
- Issue #23:** GA About To Release 3.2
- Issue #24:** Beyond The Power Of Pick, Revisited
- Issue #26:** How To Find Wasted Disk Space
- Issue #27:** The Myth Of Separation=1
- Issue #28:** Is BASIC Faster Than Access?

Although EQU statements are convenient shorthand for string and numeric constants and array references, they don't affect our compiler's code generation at all (except for the end-of-line byte after the EQU statement itself).

One exception to this rule is when EQU equates a variable name with the CHAR function. In that case, the compiler actually computes CHAR's value. For example, `PRINT CHAR(7)` generates nine bytes of code in order to compute and output the bell character at execution time. If an assignment statement is used to save the value of CHAR and allow convenient multiple references, then `BELL=CHAR(7)` requires eleven bytes, and each `PRINT BELL`: only requires four bytes. But the best approach of all is to use `EQU BELL TO CHAR(7)`, which generates no execution time code at all, and each `PRINT BELL`: still only takes four bytes of code using a compiler-computed string constant instead of a variable LOAD.

There are many other ways to make the compiler save a byte of object code here and there. For example, the statement `IF X < Y THEN PRINT "LT" ELSE PRINT "GE"` generates one less byte of object code than `IF X >= Y THEN PRINT "GE" ELSE PRINT "LT"`, even though the two statements are logically equivalent. The reason is that in order to compute greater-than-or-equal, the compiler generates code that computes less-than and negates the result, the negate taking the extra byte. Interestingly, `X > Y` generates one byte *more* than `X <= Y`! Similarly, `X # Y` generates one byte more than `X = Y`, while `IF X THEN PRINT "T" ELSE PRINT "F"` requires one less byte than `IF NOT(X) THEN PRINT "F" ELSE PRINT "T"`.

Dimensioned and dynamic arrays are another trade-off frequently considered by Pick programmers, usually because the statements differ so much in execution speed. (See *Product Profiles #21*.) We were interested to find that our compiler handles dynamic array references slightly more compactly than dimensioned array references. For example, the statement `X(I)=X(I)+1` generates 25 bytes of object code, while `X<I>=X<I>+1` requires only 23 bytes. Unfortunately, our compiler didn't do anything special for the newer syntax for extract and replace operations: `X=Y<10>` compiled exactly like `X=EXTRACT(Y,10,0,0)`, as did `X<10>=Y` and `X=REPLACE(X,10;Y)`.

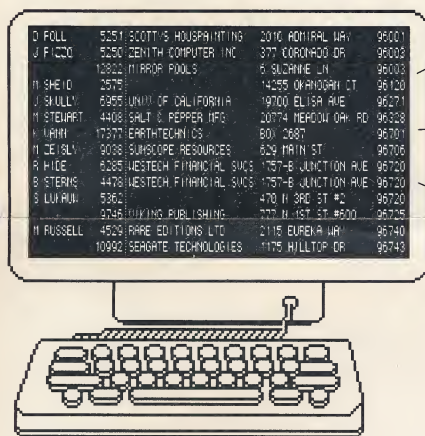
One last example is an interesting trade-off we found while comparing READV and OCONV. The statement `READV LABEL FROM ID,3 ELSE LABEL=""` generates 25 bytes of code and requires a prior OPEN statement, but `LABEL=OCONV(ID,"T<file>;X;;3")` performs the exact same function without an OPEN in only 15 bytes plus the number of characters in the string `<file>`. Unfortunately, benchmarks such as those published in *Pragma #3* have shown that the OCONV function executes almost twice as slowly as the READV statement. Δ

Tired of waiting for your
Pick™ computer to SORT or
SELECT your large data files?

Need to quickly find *any*
attribute? Want to scroll files
up *or* down, in any sort order?

Now you can use B-TREE-P to instantly search, sort, and scroll any data from any Pick file!

Now you can instantly look up customers by name, street, Zip code, or any other field — not just by customer number. Now you can immediately find inventory entries by quantity, cost, or description — not just by part number. Whatever files you use, now you can instantly locate and display your data *any way you want*, without having to wait for endless SORTs and SELECTs.



Immediately display any record in any file just by typing one or more starting characters that match *any* field in the record.

You can display not only a selected record, but also any previous and next records, using *any* sort order you specify.

You can jump to any record in a file at any time, then browse through the file by scrolling up *or* down, a record at a time, or a page at a time, in *any* sort order.

Ask us for a free copy of Product Profiles #24, describing how B-TREE-P was originally developed and put to work. As one of Semaphore's programmers says: "We often ask ourselves why we waited so long to create B-TREE-P. After using it for our own production work, we wonder how we ever got by before without it. A Pick computer without B-TREE-P is like a car without wheels".

B-TREE-P is a proven collection of BASIC subprograms for using B-trees on Pick computer systems. B-trees allow any of the data in any of your Pick files to be instantly located and displayed in any sort order, without having to wait for SORT or SELECT commands.

B-TREE-P and a few minor modifications to your existing data entry programs are all that is necessary for you to immediately be able to search, display, and browse through your data quickly and conveniently.

Modifications to your existing data files are absolutely unnecessary!



Pick is a trademark of Pick Systems.

B-TREE-P includes all necessary
BASIC source code for a B-tree
system that works with any file:

- Insertion subroutine
- Deletion subroutine
- Lookup subroutine
- Previous/next subroutine
- Complete instructions

Plus, you receive the source code
for a complete demonstration
system that uses B-TREE-P to
maintain a name and address file:

Editor program for creating and
changing names and addresses.

Browser program for displaying
names and addresses.

Printer program for listing file
items in order without having to
wait for a sort.

Here's how to order:

Send your name, address,
telephone number, and your
check for \$395 payable to
Semaphore Corporation to:

Semaphore Corporation
207 Granada Drive
Aptos, CA 95003

We'll send you complete
B-TREE-P source code
listings and all necessary
documentation.

Call us at (408) 688-9200!

WARNING: B-TREE-P includes a license
agreement with copy, use, and transfer
restrictions limiting your use of B-TREE-P to
one computer at a time. Multi-CPU and OEM
resale agreements are also available.

Made for
PICK!!



SERIAL COMMUNICATION
BOARD

STREAMING TAPE

1.2 MB
FLOPPY
DISK
DRIVE

HARD DISK

EXPANSION
BOARD SLOTS

- Altos 3068 Features!
- Microprocessor —
32 Bit MC68020
 - Cache Memory —
on chip: 256 Bytes
on board: 8 KB 2
set associative data
in instruction cache
 - Bus —
32 Bit data bus
 - Main Memory —
1-8 MB
 - Hard Disk —
Up to 195 MB
Formatted
Track caching
 - Streaming Tape —
60 MB:
90 IPS
 - Floppy Disk —
1.2 MB

ALTOS
COMPUTER SYSTEMS
WORLD LEADER IN
MULTI-USER MICROSYSTEMS

The 68020-based Altos 3068.

If you're a PICK software developer or dealer,
you owe it to yourself to see a demonstration
of the forty-user Altos 3068.
Call this number, and we'll make
the arrangements.
(800) ALTOS U.S.

PICK is a trademark of Pick Systems.

ALTOS[®]

COMPUTER SYSTEMS
2641 Orchard Parkway
San Jose, CA 95134

WE SUPPORT PICK.